Chapter 2. The UNIX Architecture and Command Usage

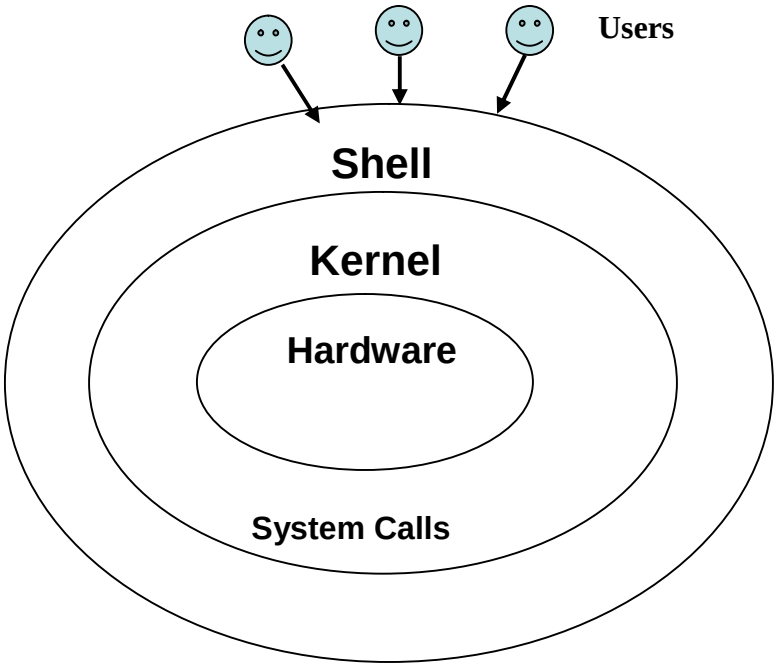
Introduction

In order to understand the subsequent chapters, we first need to understand the architecture of UNIX and the concept of division of labor between two agencies viz., the shell and the kernel. This chapter introduces the architecture of UNIX. Next we discuss the rich collection of UNIX command set, with a specific discussion of command structure and usage of UNIX commands. We also look at the man command, used for obtaining online help on any UNIX command. Sometimes the keyboard sequences don't work, in which case, you need to know what to do to fix them. Final topic of this chapter is troubleshooting some terminal problems.

Objectives

- The UNIX Architecture
- Locating Commands
- Internal and External Commands
- Command Structure and usage
- Flexibility of Command Usage
- The man Pages, apropos and whatis
- Troubleshooting the terminal problems

1. The UNIX Architecture



UNIX architecture comprises of two major components viz., the shell and the kernel. The kernel interacts with the machine's hardware and the shell with the user.

The kernel is the core of the operating system. It is a collection of routines written in C. It is loaded into memory when the system is booted and communicates directly with the hardware. User programs that need to access the hardware use the services of the kernel via use of system calls and the kernel performs the job on behalf of the user. Kernel is also responsible for managing system's memory, schedules processes, decides their priorities.

The shell performs the role of command interpreter. Even though there's only one kernel running on the system, there could be several shells in action, one for each user who's logged in. The shell is responsible for interpreting the meaning of metacharacters if any, found on the command line before dispatching the command to the kernel for execution.

The File and Proces

A file is an array of bytes that stores information. It is also related to another file in the sense that both belong to a single hierarchical directory structure.

A process is the second abstraction UNIX provides. It can be treated as a time image of an executable file. Like files, processes also belong to a hierarchical structure. We will be discussing the processes in detain in a subsequent chapter.

2. Locating Files

All UNIX commands are single words like ls, cd, cat, etc. These names are in lowercase. These commands are essentially *files* containing programs, mainly written in C. Files are stored in directories, and so are the binaries associated with these commands. You can find the location of an executable program using type command:

```
$ type ls
ls is /bin/ls
```

This means that when you execute ls command, the shell locates this file in /bin directory and makes arrangements to execute it.

The Path

The sequence of directories that the shell searches to look for a command is specified in its own PATH variable. These directories are colon separated. When you issue a command, the shell searches this list in the sequence specified to locate and execute it.

3. Internal and External Commands

Some commands are implemented as part of the shell itself rather than separate executable files. Such commands that are built-in are called internal commands. If a command exists both as an internal command of the shell as well as an external one (in /bin or /usr/bin), the shell will accord top priority to its own internal command with the same name. Some built-in commands are echo, pwd, etc.

4. Command Structure

UNIX commands take the following general form:

verb [options] [arguments]

where verb is the command name that can take a set of optional options and one or more optional arguments.

Commands, options and arguments have to be separated by spaces or tabs to enable the shell to interpret them as words. A contiguous string of spaces and tabs together is called a whitespace. The shell compresses multiple occurrences of whitespace into a single whitespace.

Options

An option is preceded by a minus sign (-) to distinguish it from filenames.

Example: $ ls –l

There must not be any whitespaces between – and l. Options are also arguments, but given a special name because they are predetermined. Options can be normally compined with only one – sign. i.e., instead of using

$ ls –l –a –t

we can as well use,

$ ls –lat

Because UNIX was developed by people who had their own ideas as to what options should look like, there will be variations in the options. Some commands use + as an option prefix instead of -.

Filename Arguments

Many UNIX commands use a filename as argument so that the command can take input from the file. If a command uses a filename as argument, it will usually be the last argument, after all options.

Example: cp file1 file2 file3 dest_dir
rm file1 file2 file3

The command with its options and argumens is known as the command line, which is considered as complete after *[Enter]* key is pressed, so that the entire line is fed to the shell as its input for interpretation and execution.

Exceptions

Some commands in UNIX like pwd do not take any options and arguments. Some commands like who may or may not be specified with arguments. The ls command can run without arguments (ls), with only options (ls –l), with only filenames (ls f1 f2), or using a combination of both (ls –l f1 f2). Some commands compulsorily take options (cut). Some commands like grep, sed can take an expression as an argument, or a set of instructions as argument.

5. Flexibility of Command Usage

UNIX provides flexibility in using the commands. The following discussion looks at how permissive the shell can be to the command usage.

Combining Commands

Instead of executing commands on separate lines, where each command is processed and executed before the next could be entered, UNIX allows you to specify more than one command in the single command line. Each command has to be separated from the other by a ; (semicolon).

```
wc sample.txt ; ls –l sample.txt
```

You can even group several commands together so that their combined output is redirected to a file.

```
(wc sample.txt ; ls –l sample.txt) > newfile
```

When a command line contains a semicolon, the shell understands that the command on each side of it needs to be processed separately. Here ; is known as a metacharacter.

Note: When a command overflows into the next line or needs to be split into multiple lines, just press enter, so that the secondary prompt (normally >) is displayed and you can enter the remaining part of the command on the next line.

Entering a Command before previous command has finished

You need not have to wait for the previous command to finish before you can enter the next command. Subsequent commands entered at the keyboard are stored in a buffer (a temporary storage in memory) that is maintained by the kernel for all keyboard input. The next command will be passed on to the shell for interpretation after the previous command has completed its execution.

6. man: Browsing The Manual Pages Online

UNIX commands are rather cryptic. When you don't remember what options are supported by a command or what its syntax is, you can always view man (short for manual) pages to get online help. The man command displays online documentation of a specified command.

A pager is a program that displays one screenful information and pauses for the user to view the contents. The user can make use of internal commands of the pager to scroll up and scroll down the information. The two popular pagers are more and less. more is the Berkeley's pager, which is a superior alternative to original pg command. less is the standard pager used on Linux systems. less if modeled after a popular editor called vi and is more powerful than more as it provides vi-like navigational and search facilities. We can use pagers with commands like ls | more. The man command is configured to work with a pager.

7. Understanding The man Documentation

The man documentation is organized in eight (08) sections. Later enhancements have added subsections like 1C, 1M, 3N etc.) References to other sections are reflected as SEE ALSO section of a man page.

When you use man command, it starts searching the manuals starting from section 1. If it locates a keyword in one section, it won't continue the search, even if the keyword occurs

in another section. However, we can provide the section number additionally as argument for man command.

For example, passwd appears in section 1 and section 4. If we want to get documentation of passwd in section 4, we use,

$ man 4 passwd OR $ man –s4 passwd (on Solaris)

Understanding a man Page

A typical man page for wc command is shown below:

```
User Commands                          wc(1)
NAME
     wc – displays a count of lines, words and characters
in a file
SYNOPSIS
     wc [-c | -m | -C] [-lw] [file ...]
DESCRIPTION
     The wc utility reads one or more input files and, by
default, writes the      number of newline characters,
words and bytes  contained in each input file     to the
standard output. The utility also writes a total count
for all named  files, if more than one input file is
specified.
OPTIONS
     The following options are supported:
     -c   Count bytes.
     -m   Count characters.
     -C   same as –m.
     -l   Count lines.
     -w   Count words delimited by white spaces or new
line characters ...
OPERANDS
     The following operand is supported:
     file A path name of an input file. If no file
operands are specified,       the standard input will be
used.
EXIT STATUS
     See largefile(5) for the description of the behavior
of wc when     encountering files greater than or equal
to 2 Gbyte (2 **31 bytes)
SEE ALSO
     cksum(1), isspace(3C), iswalpha(3C), iswspace(3C),
largefile(5), ...
```

A man page is divided into a number of compulsory and optional sections. Every command doesn’t need all sections, but the first three (NAME, SYNOPSIS and

DESCRIPTION) are generally seen in all man pages. NAME presents a one-line introduction of the command. SYNOPSIS shows the syntax used by the command and DESCRIPTION provides a detailed description.

The SYNOPSIS follows certain conventions and rules:

- If a command argument is enclosed in rectangular brackets, then it is optional; otherwise, the argument is required.
- The ellipsis (a set if three dots) implies that there can be more instances of the preceding word.
- The | means that only one of the options shows on either side of the pipe can be used.

All the options used by the command are listed in OPTIONS section. There is a separate section named EXIT STATUS which lists possible error conditions and their numeric representation.

Note: You can use man command to view its own documentation ($ man man). You can also set the pager to use with man ($ PAGER=less ; export PAGER). To understand which pager is being used by man, use $ echo $PAGER.

The following table shows the organization of man documentation.

Section	Subject (SVR4)	Subject (Linux)
1	User programs	User programs
2	Kernel's system calls	Kernel's system calls
3	Library functions	Library functions
4	Administrative file formats	Special files (in /dev)
5	Miscellaneous	Administrative file formats
6	Games	Games
7	Special files (in /dev)	Macro packages and conventions
8	Administration commands	Administration commands

8. Further Help with man –k, apropos and whatis

man –k: Searches a summary database and prints one-line description of the command.
Example:

```
$ man –k awk
awk  awk(1)    -pattern scanning and processing language
nawk nawk(1)   -pattern scanning and processing language
```

apropos: lists the commands and files associated with a keyword.
Example:

```
$ apropos FTP
ftp  ftp(1)         -file transfer program
ftpd in.ftpd(1m)    -file transfer protocol server
ftpusers  ftpusers(4) -file listing users to be disallowed
                        ftp login privileges
```

whatis: lists one-liners for a command.

Example:

```
$ whatis cp
cp   cp(1)          -copy files
```

9. When Things Go Wrong

Terminals and keyboards have no uniform behavioral pattern. Terminal settings directly impact the keyboard operation. If you observe a different behavior from that expected, when you press certain keystrokes, it means that the terminal settings are different. In such cases, you should know which keys to press to get the required behavior. The following table lists keyboard commands to try when things go wrong.

Keystroke or command	**Function**
[Ctrl-h]	Erases text
[Ctrl-c] or Delete	Interrupts a command
[Ctrl-d]	Terminates login session or a program that expects its input from keyboard
[Ctrl-s]	Stops scrolling of screen output and locks keyboard
[Ctrl-q]	Resumes scrolling of screen output and unlocks keyboard
[Ctrl-u]	Kills command line without executing it
[Ctrl-\]	Kills running program but creates a core file containing the memory image of the program
[Ctrl-z]	Suspends process and returns shell prompt; use **fg** to resume job
[Ctrl-j]	Alternative to *[Enter]*
[Ctrl-m]	Alternative to *[Enter]*
stty sane	Restores terminal to normal status

Conclusion

In this chapter, we looked at the architecture of UNIX and the division of labor between two agencies viz., the shell and the kernel. We also looked at the structure and usage of UNIX commands. The man documentation will be the most valuable source of documentation for UNIX commands. Also, when the keyboard sequences won't sometimes work as expected because of different terminal settings. We listed the possible remedial keyboard sequences when that happens.